Evidence that Incremental Delta-Bar-Delta Is an Attribute-Efficient Linear Learner

Harlan D. Harris

University of Illinois at Urbana-Champaign, Department of Computer Science
MC-258, Urbana, IL 61801 USA
hharris@uiuc.edu

Abstract. The Winnow class of on-line linear learning algorithms [10,11] was designed to be attribute-efficient. When learning with many irrelevant attributes, Winnow makes a number of errors that is only logarithmic in the number of total attributes, compared to the Perceptron algorithm, which makes a nearly linear number of errors. This paper presents data that argues that the Incremental Delta-Bar-Delta (IDBD) second-order gradient-descent algorithm [14] is attribute-efficient, performs similarly to Winnow on tasks with many irrelevant attributes, and also does better than Winnow on a task where Winnow does poorly. Preliminary analysis supports this empirical claim by showing that IDBD, like Winnow and other attribute-efficient algorithms, and unlike the Perceptron algorithm, has weights that can grow exponentially quickly. By virtue of its more flexible approach to weight updates, however, IDBD may be a more practically useful learning algorithm than Winnow.

1 Introduction

Linear learning algorithms make predictions by computing linear functions of their inputs. Since linear learners aren't capable of representing or learning non-linear concepts, practical use often requires either layering multiple linear learners and using a backpropagation algorithm, or generating an expanded feature space. Expanded feature spaces typically involve generating combinations of the original features, resulting in very large numbers of attributes irrelevant to the concept to be learned. The two algorithms most studied for use with this approach have been the Perceptron algorithm and Winnow [10,8,12,7]. Some learning domains such as computer vision and natural language processing naturally provide very large feature spaces with many irrelevant attributes, even without an expanded feature space. In this paper, I provide evidence that the Incremental Delta-Bar-Delta (IDBD) algorithm [14] combines the attribute-efficient properties of Winnow with additional robustness and flexibility, and is particularly useful for learning when many attributes may be irrelevant.

In the on-line learning framework, the learner repeatedly performs a prediction task and receives a supervised training signal. Examples $\mathbf{x}$ are selected from an instance space, and are labeled by a concept c in a concept space. For each trial, the learner is given the example, predicts the example's label, e.g.

T. Elomaa et al. (Eds.): ECML, LNAI 2430, pp. 135–147, 2002.

$p \in \{-1, 1\}$, and then receives the true label, e.g. $\ell \in \{-1, 1\}$, where $\ell = c(\mathbf{x})$. The goal of an on-line learner is to minimize the total number of mistakes ($\ell \neq p$) made in the prediction task while learning concept c.

For a linear learner, the concept space is linear functions, as represented by a linear threshold unit, or Perceptron. Given an input vector $\mathbf{x}$, of width n, a Perceptron computes the function $p = \text{sign}(\mathbf{w} \cdot \mathbf{x})$, where $\mathbf{w}$ is the weight vector of the Perceptron. Since the hyperplane defined by the weight vector alone always includes the zero, Perceptrons frequently include a fixed or trainable bias weight.

In order to learn non-linearly-separable functions, a common approach is to generate conjunctions of the inputs, either explicitly [12], or using kernel functions [2]. Although arbitrary DNF expressions can then be represented, linear functions being adequate to represent disjunctions of these conjunctions, several issues remain. There are an exponential number of conjunctions of the n input features, requiring exponential time and space to process. Use of kernel functions present other problems, particularly in domains where the input space is naturally large and mostly irrelevant [7]. In perhaps the most important problem for on-line learning, the number of examples needed to learn may increase linearly with the size of the expanded input space. An attribute-efficient algorithm is one in which the number of required examples increases only logarithmically in the number of irrelevant features.

In an engineering setting, these expanded feature space approaches have been used most commonly in natural language processing (e.g., [12]), in which the presence or absence of particular words or word combinations leads to very large numbers of sparse, irrelevant features. Valiant [15] motivates this type of expanded-basis learning by reviewing biological systems, which are able to learn quickly in settings with very large numbers of interconnected neurons. For example, in the cerebellum, the neuronal architecture closely resembles the expanded feature space and linear learner paradigm [5,13].

2 Algorithms

2.1 Perceptron Learning Rule

The Perceptron learning rule trains a linear threshold unit (Perceptron) by adding a fraction of each mis-predicted input vector to the weight vector.

$$w_i \leftarrow w_i + \eta(\ell - p)x_i, \tag{1}$$

where $\eta \in (0, 1)$ is a learning rate parameter. The Perceptron rule performs incremental gradient descent in weight space. Variations include the Least-Mean-Square (LMS) rule (for non-thresholded linear units), and backpropagation (for multi-layer networks).

2.2 Winnow

The Winnow algorithms were introduced by Littlestone [10,11] as linear learners that separate (winnow) relevant from irrelevant attributes. Winnow was designed

to efficiently learn monotone (non-negative) disjunctions and r-of-k threshold functions, and many of its proofs and applications have been in those domains.

There are a number of variations of Winnow, optimized for different domains and with varying notation. The Winnow2 version of the algorithm, shown below, is typically initialized with all weights set to 1 or $\frac{1}{n}$. The threshold is often set to n, giving a ratio of threshold to initial weights of $n:1$ or $n^2:1$.

$$w_i \leftarrow \begin{cases} w_i \alpha^{x_i} & \text{if } p = -1 \wedge \ell = 1, \\ w_i (1/\alpha)^{x_i} & \text{if } p = 1 \wedge \ell = -1, \\ w_i & \text{otherwise,} \end{cases} \tag{2}$$

where $x_i \in \{0, 1\}$. The Balanced Winnow variation learns non-monotone functions.

Winnow works by forcing weights on irrelevant features towards zero exponentially quickly for false positives, and by raising weights on relevant features exponentially quickly for false negatives. The use of a multiplicative rather than additive update rule means that fewer changes are needed for convergence.

The mistake bounds for the Winnow and Perceptron algorithms have been closely compared [8]. When learning monotone disjunctions of r literals, Winnow makes no more than $O(r \log n)$ mistakes, where n is the total number of attributes, and is thus attribute-efficient. In contrast, when learning the same class of functions, the Perceptron learning rule makes a nearly linear (in n) number of mistakes. If n is much larger than r, then Winnow performs much better, while if n is not much larger then r, the Perceptron algorithm does better, particularly if few features are active (non-zero) at any one time.

2.3 IDBD

Sutton's IDBD (Incremental Delta-Bar-Delta) algorithm [14] is a variation on Jacobs' Delta-Bar-Delta algorithm [6]. Delta-Bar-Delta, in turn, is a variation on the LMS learning rule (and also on backpropagation) that includes heuristics designed to speed up learning, most notably the inclusion of per-weight learning rates. The incremental version, IDBD, is additionally suitable for on-line learning and for learning non-stationary concepts. (Unlike DBD, IDBD is restricted to learning weights for linear units, and no backpropagation variant has apparently been derived. Note also that IDBD was originally derived to learn linear functions, with real-valued inputs and non-thresholded outputs, but I am using it here to learn weights for a linear-threshold unit and Boolean inputs.) As shown below, the particular way that IDBD accelerates learning seems to put it in the same category of attribute-efficient learning algorithms as Winnow.

IDBD uses an additive weight update rule with a per-unit modifiable learning rate, rather than a shared, fixed learning rate as in the Perceptron rule. The algorithm can be seen to be performing gradient descent in learning-rate space, as well as in weight-space [14]. Intuitively, if a weight repeatedly changes in the same direction, then the learning rate is increased, since this input's weight appears not to have converged (it is either too high or too low). If the weight

changes appear to be random, then the learning rate is decreased, since this input's weight appears to have converged, and the weight is oscillating around the correct value. The update rules are as follows:

$$\beta_i = \beta_i + \theta h_i x_i (\ell - p) \tag{3}$$

$$\eta_i = e^{\beta_i} \tag{4}$$

$$w_i = w_i + \eta_i x_i (\ell - p) \tag{5}$$

$$h_i = h_i [1 - \eta_i x_i^2]^+ + \eta_i x_i (\ell - p) \tag{6}$$

Note the per-input learning rate, η_i, in equation 5. Also, observe that β is updated using the Perceptron rule, using θ as a learning rate, with the addition of h_i, which represents the recent summed history of the weight's changes. The $[\cdot]^+$ notation is equivalent to $max(\cdot, 0)$. An interesting aspect of IDBD is that its update rules are defined recursively; β depends on h, which depends on β via η.

Sutton [14] gives experimental results showing that IDBD has lower cumulative error on a simple non-stationary tracking task (where the concept periodically changes) than does the LMS rule, is able to distinguish relevant from irrelevant inputs, and weights those inputs approximately optimally. However, he does not address the issue of attribute-efficiency, and no further theoretical or empirical examination of IDBD has been performed to my knowledge.

3 Experiments

In this section I present new experimental evidence that compares IDBD to Winnow and the Perceptron algorithm. Experiment 1 tests these algorithms on the l-of-m-of-n threshold function, and shows that IDBD usually performs comparably to Winnow, and much better than the Perceptron learning rule, when learning disjunctive concepts with many irrelevant attributes. Experiment 2 systematically varies the complexity parameter n, with results suggesting that IDBD's mistake bounds grow logarithmically, like Winnow, and unlike the Perceptron algorithm. Experiment 2b uses the same domain to illustrate IDBD's low need for parameter tuning. Experiment 3 then looks at learning of random linear functions with few irrelevant attributes, and shows that IDBD performs well even in circumstances when Winnow does poorly, which reinforces the idea that IDBD is not merely a reimplementation of Winnow. (In an earlier experiment, IDBD was shown to be useful for learning complex DNF concepts with incomplete expanded feature spaces [5].)

For all results reported below, 10 replications were performed with different random number seeds used to generate the data, and the results were averaged. Initial weights for all three algorithms were set to $\frac{1}{n}$, and the thresholds were $\frac{n}{2}$.

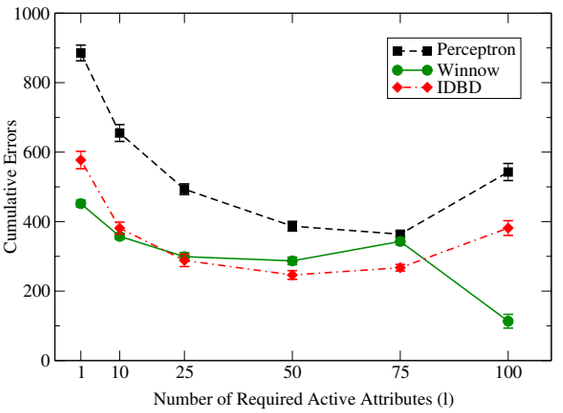

Fig. 1. Cumulative errors after 10,000 examples in an l-of-100-of-1000 learning task. Mean of ten replications, with 95% confidence intervals. Winnow performs best when $l = 1$ (disjunction) or $l = 100$ (conjunction), but IDBD performs similarly or better for intermediate values of l

3.1 Experiment 1: Comparing IDBD to Winnow with Irrelevant Attributes

This experiment was designed to compare IDBD to Winnow and the Perceptron algorithm on a task covered by the theoretical results of Kivinen et al. [8]. When learning a concept where the Perceptron rule should make a nearly linear (in irrelevant attributes) number of errors, and Winnow should make a logarithmic (in irrelevant attributes) number of errors, how does IDBD empirically perform?

To test this, the algorithms were compared on l-of-m-of-n Boolean concepts. These concepts are defined so that if l or more of the first m attributes are set to 1, the example is positive, and if fewer than l of the first m attributes are set to 1, the example is negative. Data was generated as follows. Each input vector was of width n. The first m attributes were considered "relevant," while the remaining $n - m$ attributes were "irrelevant," and were set to 1 with a probability of 0.25. The examples were half positive and half negative. For positive examples, a number r in the interval $[l, m]$ was chosen, while for negative examples, r was in the interval $[0, l - 1]$. Then, r attributes in the first m were randomly selected and were set to 1, with the other relevant attributes set to 0. 10,000 examples were generated and presented to the algorithms.

Based on pilot experiments, the learning rates of the Perceptron rule (η) and IDBD (θ) were set to 0.1, and the learning rate of Winnow (α) was set to 1.1.

Table 1. Error rate in final 1000 examples of an l-of-100-of-1000 learning task. Mean of ten replications

	1	10	25	50	75	100
Perceptron	6.8	5.0	3.8	3.0	2.9	4.8
Winnow	1.6	1.0	1.1	1.4	2.6	0.0
IDBD	4.1	2.7	2.1	1.7	2.1	3.2

Moderate changes to these rates do not qualitatively change the results (see also Experiment 2b).

Each algorithm was run with various values of l, with $m = 100$ and $n = 1000$. The number of errors after presentation of 10,000 examples is shown in Figure 1. The error rate for the last 1000 examples is shown in Table 3.1. IDBD performed uniformly much better than Perceptron, with fewer cumulative errors and lower final error rates. Compared to Winnow, IDBD usually made a similar number of cumulative errors, but often with a somewhat higher residual error rate.

3.2 Experiment 2: Showing Attribute-Efficient Learning by IDBD

The theoretical results predict that as n grows, the Perceptron algorithm should make errors that increase nearly linearly, while Winnow's mistakes should increase only logarithmically. If IDBD is attribute-efficient, its results should be similar to Winnow's.

To test this, l and m were set to 10 and 100, respectively, and n was varied between 100 and 1000. Instead of running for a fixed number of examples, each trial was terminated when 200 examples in a row were classified correctly (i.e., the concept had been effectively learned). The learning rates for the Perceptron rule (η), IDBD (θ), and Winnow (α) were set to 0.8, 0.1, and 1.4, respectively (see below). This test shows precisely how well each algorithm scales up with irrelevant attributes, and should illustrate the theoretical predictions discussed above.

The results are shown in Figure 2. The Perceptron algorithm, as expected, did much more poorly than Winnow, and its mistake measures increased sharply with n. Winnow made only slightly more errors as n increased, and the curve appears logarithmic.

Most interestingly, IDBD shares Winnow's property of making few additional errors as n increases. Like Winnow, IDBD's attribute-efficiency curve seems logarithmic, and is qualitatively different from the Perceptron algorithm. These results strongly suggest that IDBD shares the irrelevant-attribute-efficiency that Winnow is know for.

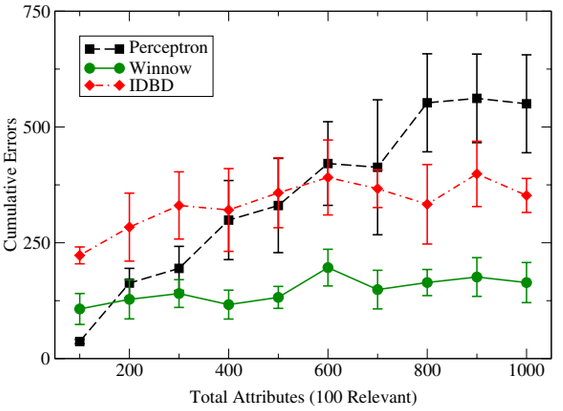

Fig. 2. Cumulative errors at convergence (200 correct predictions in a row) for a 10-of-100-of-n learning task. Mean of ten replications, with 95% confidence intervals. IDBD and Winnow show attribute efficiency, unlike the Perceptron algorithm

3.3 Experiment 2b: Showing IDBD Is Relatively Insensitive to Its Parameters

To more carefully explore the strengths and weaknesses of these algorithms, and to replicate one of Sutton's conclusions about IDBD [14], the same domain was used for an exploration of the learning rate parameter space. As before, the target concepts are 10-of-100-of-n threshold functions. The learning rates tested were 0.001, 0.01, 0.05, 0.1, 0.2, 0.4, and 0.8. (For Winnow, α was set to 1 plus the above numbers.) The variable n took the values 100, 500, and 1000. As before, 10 replications were performed, and the cumulative errors before convergence (200 correct predictions in a row) were counted.

The results are shown in Table 3.3. Clearly, the IDBD algorithm was relatively insensitive to small values of the learning rate, as compared with Winnow and the Perceptron algorithm. However, it was susceptible to convergence failure when the learning rate was very high. Again, note the sharp increase in errors between $n = 500$ and $n = 1000$ for the Perceptron algorithm, compared to the small increases for Winnow and IDBD.

3.4 Experiment 3: Showing IDBD Does Well when Winnow Does Poorly

The same theoretical results that predict that Winnow should learn with fewer mistakes when irrelevant attributes are predominant also predict that the Perceptron learning rule should learn with fewer mistakes when all (or nearly all) attributes are relevant. In addition, since Winnow is optimized for disjunctions, it's reasonable to expect that it will do relatively poorly when learning arbitrary weights. By virtue of its fixed, multiplicative weight update rule, Winnow may find it difficult to set weights with the precision needed for arbitrary, small-margin concepts. Weights can oscillate around a target value, their step size too large to approach it. To test this intuition, the three algorithms were compared on randomly-generated linear threshold functions. These functions are of the form $c(\mathbf{x}) = \tilde{\mathbf{w}} \cdot \mathbf{x}$, where the elements of $\tilde{\mathbf{w}}$ were randomly generated, independent real-valued numbers.

The data was generated as follows. For each concept, target weights were selected from a uniform distribution over the interval $[0, 10]$. Then, 20,000 examples of each concept were generated by randomly setting Boolean inputs with probability 0.5, multiplying by the target concept's weights, and comparing the result to a threshold equal to the expected value, $10 * \frac{n}{4}$. Approximately half of the resulting examples were thus positive, and half were negative. Note that weights near 10 are maximally relevant, while those near zero are essentially irrelevant. Unlike the previous experiments, there is a continuum of relevance, with nearly all weights being somewhat relevant to the target concept.

In this experiment, we used learning rates of 0.01 for the Perceptron rule, 0.2 for IDBD, and 1.01 for Winnow, based on informal pilot experiments.

The results, for $n = 200$ (other values of n were similar), can be seen in Figure 3. As expected, the Perceptron algorithm performed better than Winnow.

Table 2. Cumulative errors to convergence on a 10-of-100-of-n task, varying learning rates. Bold represent the parameter that results in the lowest error count at $n = 1000$. *NC* means that the algorithm did not always converge even after 100,000 examples

Alg.	n	.001	.01	.05	.1	.2	.4	.8
Percep.	100	11645	1671	382	216	115	63	**37**
	500	12495	2889	1215	823	434	413	**330**
	1000	14608	4800	1415	1018	635	562	**550**
Winnow	100	15145	1667	358	190	115	**107**	192
	500	20442	2126	445	240	153	**132**	278
	1000	22762	2390	502	261	185	**164**	480
IDBD	100	380	365	289	**223**	171	118	180
	500	1185	910	421	**358**	211	*NC*	*NC*
	1000	1374	1097	553	**352**	*NC*	*NC*	*NC*

The IDBD algorithm showed performance similar to, and slightly but significantly better than, that of the Perceptron learning rule for this domain.

4 Discussion

These empirical results suggest that IDBD is an attribute-efficient learner, with a logarithmic attribute-efficiency curve. IDBD makes mistakes at rates only somewhat higher than Winnow when irrelevant attributes are plentiful, but significantly lower than Winnow when all attributes are relevant. By being both attribute-efficient and flexible, IDBD should be particularly useful when the number of irrelevant attributes is unknown prior to learning. For example, robotic multi-modal sensory systems, computational linguistic applications [12], and neurological modeling applications [5,15] often naturally have extensive irrelevant attributes and expanded feature spaces.

Other linear learning systems with attribute-efficient properties have been mentioned in the literature. The p-norm family of Generalized Perceptron[1] algorithms [4] is able to approximate the Perceptron and Winnow algorithms by choices of the p parameter. For a sufficiently large p, the p-norm algorithm is known to be attribute-efficient. The ALMA$_p$ algorithm [3] combines a p-norm-like computation with a decaying learning rate, a specifiable margin, and nor-

[1] The Generalized Perceptron defines Quasi-Additive algorithms defined by a vector $\mathbf{z}$, updated exactly like the Perceptron, and a function $\mathbf{f}(\mathbf{z}) = \mathbf{w}$ specifying the weight vector as a function of $\mathbf{z}$. The Perceptron, Balanced Winnow, and p-norm algorithms may be defined by $\mathbf{f}(\mathbf{z}) = \mathbf{z}$, $\sinh(\mathbf{z})$, and $\text{sign}(\mathbf{z})p|\mathbf{z}|^{p-1}$, respectively.

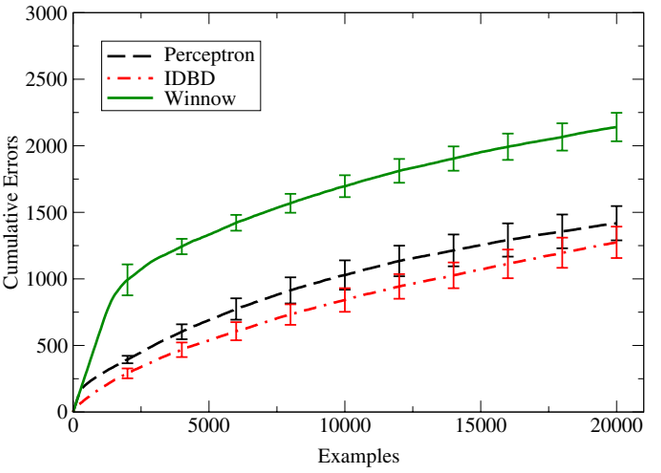

Fig. 3. Cumulative errors on 200-attribute random linear threshold concepts. Average of ten replications, with 95% confidence intervals

malized weight vectors. Although it has not been shown to be attribute-efficient, it seems likely to be so.

IDBD has similarities with these two algorithms, but differences and advantages as well. Like p-norm and Winnow, and unlike ALMA$_p$, IDBD's weights are unbounded, and by having increasing learning rates[2], can grow exponentially fast. Like ALMA$_p$, but unlike p-norm and Winnow, IDBD uses adjustable learning rates which decrease as the learner converges. However, IDBD has per-weight learning rates, while ALMA$_p$ has only a single dynamic learning rate.

A practical advantage in real-world learning situations is that IDBD has only a single parameter, the learning rate θ, compared to p-norm with two and ALMA$_p$ with three. (IDBD has more variables requiring initial conditions, however.) IDBD is relatively insensitive to the settings of its learning rate parameter [14], allowing IDBD to be used with more confidence given less knowledge of the target concept than other algorithms. A final advantage is that IDBD was designed to be capable of learning drifting concepts, while other attribute-efficient learners may deal poorly with concept change. (But see [1] for a variation on Winnow which does learn drifting concepts well.)

4.1 Analytical Support

Several analytical approaches can be used to complement the empirically-supported assertion that IDBD is attribute-efficient. In this section, I investigate the relative speed of weight changes in various algorithms, and argue that exponential weight increases are sufficient for attribute-efficiency. Then, I show that a variation of IDBD is very similar to the Generalized Perceptron formulation of an algorithm that is known to be attribute-efficient.

IDBD's attribute-efficiency seems to be due to the algorithm's ability to increase weights exponentially fast on relevant attributes, relative to the other weights. With exponentially fast increases, relevant weights outweigh irrelevant weights quickly enough that a number of errors only logarithmic in the number of irrelevant weights need be made [10]. Consider a simple case of the weight on a relevant Boolean input, where the learner makes repeated false negative predictions. That is, for simplicity, let $x = 1$ and $\ell - p = 1$ (actually $= 2$, but we can use 1 by doubling the learning rate). How does w change over time?

For the Perceptron algorithm in this case, $w \leftarrow w + \eta$, and $w(n) = \sum_{i=1}^{n} \eta = O(n)$. The Perceptron algorithm only increases weights linearly quickly, and it is not attribute-efficient.

For Winnow under these assumptions, $w \leftarrow \alpha w$, and $w(n) = w(0)\alpha^n = O(\alpha^n)$. Clearly, Winnow increases weights exponentially quickly, and it is attribute-efficient.

Before examining IDBD, let's first examine a simple second-order gradient-descent relative, in which $\eta = \beta$ and $h = 1$. Then, (using T instead of θ so as not to be confused with the $\theta(\cdot)$ of asymptotic notation) we have $\beta \leftarrow \beta + T$

[2] Winnow can be rewritten as an additive learning rule with a per-weight learning rate that is equal to a function of the weight itself.

Table 3. IDBD-h algorithm

prediction	label		update	
1	-1	if $x_i = 1$,	$w_i \leftarrow w_i - e^{z_i}$,	demotion
			$z_i \leftarrow z_i - \eta$	
-1	1	if $x_i = 1$,	$z_i \leftarrow z_i + \eta$	promotion
			$w_i \leftarrow w_i + e^{z_i}$,	

and $w \leftarrow w + \beta$. Therefore, $\beta(n) = \sum_{i=1}^{n} T = O(n)$, and $w(n) = \sum_{i=1}^{n} \beta(n) = \sum_{i=1}^{n} \sum_{j=1}^{n} T = O(n^2)$. This algorithm does not grow weights exponentially, and thus could not be expected to show the logarithmic attribute-efficient behavior of Winnow[3].

Adding $\eta = e^{\beta}$ back to IDBD, but keeping $h = 1$, we now have that $w \leftarrow w + e^{\beta}$, and $w(n) = \sum_{i=1}^{n} e^{O(n)} = O(e^n)$. This algorithm, without the decaying sum of recent weight changes, can increase weights exponentially, and thus should be attribute-efficient. It is rather similar to Winnow, in that the effective learning rate is related to all previous weight changes.

Re-adding the h of equation 6 to complete IDBD results in an attribute-efficient algorithm that modulates the exponential in the exponentially-increasing learning rate by the extent to which the recent weight changes have been in the same direction. The result is more flexible than Winnow, yet still appears attribute-efficient.

With some modifications, IDBD can be shown to be similar to the Weighted Majority algorithm [11,9]. To see this, first modify IDBD by separating the promotion (false negative) and demotion (false positive) cases, as in the original presentation of Winnow [10]. Then, let $h_i = 1$ as above, and assume that $x_i \in \{0, 1\}$. We then have the algorithm shown in Table 3, which I'll call IDBD-h. Note that by reversing the order of w_i and z_i updates in demotions, that a sequence of promotions and demotions can be re-ordered arbitrarily and will result in the same final weight vectors.

Assuming the initial values of $\mathbf{z}$ and $\mathbf{w}$ are zero, the weight vector $\mathbf{w}$ can be written as a function of the current value of $\mathbf{z}$: $w_i = \sum_{i=1}^{z_i/\eta} e^{i\eta} = \frac{e^{\eta}}{e^{\eta}-1}(e^{z_i} - 1)$. Note that the ratio is a constant, and thus can be shifted to the bias weight, if their is one, or ignored otherwise. We therefore have a Generalized Perceptron with $f(\mathbf{z}) = e^{\mathbf{z}} - 1$. This is very similar to the Weighted Majority algorithm [9], which can be defined as a G.P. with $f(\mathbf{z}) = e^{\mathbf{z}}$ (plus simple transformations of the input representation and learning rate)[4]. Clearly, the IDBD-h algorithm should have mistake bounds that are similar to those of the attribute-efficient (see [11]) Weighted Majority. Unfortunately, direct analysis of IDBD-h has so far failed, as the methods for finding mistake bounds for Generalized Perceptron can't be easily applied to the $f(\mathbf{z})$ above, and other approaches have yet to be successful.

[3] It's interesting to consider whether this algorithm might have $O(\sqrt{n})$ mistake bounds. In fact, work in progress shows that a closely related algorithm does.

However, this analysis on a simplified version of IDBD further provides evidence for the notion of IDBD's attribute-efficiency.

4.2 Conclusions

This paper has provided an empirical basis for believing that the IDBD algorithm has attribute-efficient properties. In addition, it was shown that IDBD has strengths relative to more traditional attribute-efficient learners such as Winnow, and that analytical approaches to establishing IDBD's attribute efficiency show promise. Future work will analytically explore the space of attribute-efficient linear learners, including IDBD and related algorithms. By more clearly identifying the relationships between attribute-efficient algorithms, and by defining the functional features of each, it will become easier to explain experimental results, and to apply these algorithms to real-world problems.

Acknowledgments

I wish to thank Dan Roth, Gary Dell, Jerry DeJong, Sylvian Ray, Jesse Reichler, Dav Zimak, Ashutosh Garg, and several anonymous reviewers for their helpful suggestions on this and earlier versions of this paper. This work was in part supported by NSF grant SBR-98-73450 and NIH grant DC-00191.

References

1. P. Auer and M. K. Warmuth. Tracking the best disjunction. *Machine Learning*, 32:127–150, 1998. 144
2. N. Christiani and J. Shawe-Taylor. *An Introduction to Support Vector Machines*. Cambridge University Press, 2000. 136
3. Claudio Gentile. A new approximate maximial margin classification algorithm. *Journal of Machine Learning Research*, 2:213–242, December 2001. 143
4. Adam J. Grove, Nick Littlestone, and Dale Schurrmans. General convegence results for linear discriminant updates. *Machine Learning*, 43(3):173–210, 2001. 143, 145
5. Harlan D. Harris and Jesse A. Reichler. Learning in the cerebellum with sparse conjunctions and linear separator algorithms. In *Proceedings of the International Joint Conference on Neural Networks 2001*, 2001. 136, 138, 143
6. Robert A. Jacobs. Increased rates of convergence through learning rate adaptation. *Neural Networks*, 1:295–307, 1988. 137
7. Roni Khardon, Dan Roth, and Rocco Servedio. Efficiency versus convergence of boolean kernels for on-line learning algorithms. In *Proceedings of Neural Information Processing Systems 2001*, 2001. 135, 136
8. J. Kivinen, M. K. Warmuth, and P. Auer. The Perceptron algorithm versus Winnow: Linear versus logarithmic mistake bounds when few input variables are relevant. *Artificial Intelligence*, 97:325–343, 1997. 135, 137, 139
9. N. Littlestone and M. K. Warmuth. The weighted majority algorithm. *Information and Computation*, 108:212–261, 1994. 145
10. Nick Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Machine Learning*, 2:285–318, 1988. 135, 136, 144, 145

11. Nick Littlestone. Mistake bounds and logarithmic linear-threshold learning algorithms. PhD thesis, University of California, Santa Cruz, Technical Report UCSC-CRL-89-11, March 1989. 135, 136, 145
12. Dan Roth. Learning to resolve natural language ambiguities: A unified approach. In *Proceedings of AAAI-98, 15th Conference of the American Association for Artificial Intelligence*, pages 806–813, 1998. 135, 136, 143
13. Nicolas Schweighofer and Michael A. Arbib. A model of cerebellar metaplasticity. *Learning and Memory*, 4:421–428, 1998. 136
14. Richard S. Sutton. Adapting bias by gradient descent: An incremental version of Delta-Bar-Delta. In *Proceedings of the Tenth National Conference on Artificial Intelligence*, pages 171–176. MIT Press, 1992. 135, 137, 138, 141, 144
15. Leslie G. Valiant. Projection learning. *Machine Learning*, 37:115–130, 1999. 136, 143